

Upotreba Selenium biblioteka i alata

U okviru ove lekcije govorićemo o alatu Selenium. Objasnićemo njegovu svrhu, kako se koristi i kako postizemo testiranja pomoću ovog moćnog alata, koji je vrlo brzo postao industrijski standard u oblasti quality assurance testiranja.

Selenium je softver za upravljanje automatizacijom korišćenja aplikacije – dakle, pruža nam način da simuliramo i automatizujemo kretanje korisnika kroz web aplikacije: operacije poput klika na linkove, unosa nekih podataka i login i register akcija unutar samog pregledača.

Umesto da se operacije obavljaju ručno, koristimo biblioteku koja ovo obavlja za nas i kreiramo sistem da zabeležimo probleme u radu web aplikacije. Kao i JMeter, Selenium se koristi isključivo u okviru oblasti QA, quality assurance. Naravno, ova dva alata se mogu koristiti i uporedo, jer je fokus JMeter alata samo stres i load testiranje, dok Selenium ima funkcionalni karakter.



Slika 14.1. Selenium logo

Najveća prednost Selenium biblioteke je to što podržava veliki broj programskih jezika; trenutno postoji podrška za Javu, C#, PHP, Python, Perl, JavaScript i Ruby. Zbog podrške za najveće programske jezike, njegova primena je postala standard u ovoj oblasti. Za razliku od JMetera, gde najviše interakcije imamo sa samim interfejsom programa, usled prirode testiranja koje se obavlja u okviru Seleniuma, moramo koristiti neki od programskih jezika da bismo pisali testove. U slučaju Pythona, sve operacije navodimo u okviru jednog .py fajla koji pokreće biblioteku. U nastavku prikazujemo postupak instalacije i podešavanja Seleniuma u okviru Pythona.

Instalacija biblioteke – Selenium Web Driver

Selenium omogućava pisanje i izvršavanje funkcionalnih testova. Kako se funkcionalni testovi odlikuju direktnim korišćenjem samog programa, i sam Selenium je nastao iz potrebe da se proces ponavljanja testiranja neke funkcionalnosti automatizuje. Razlog je jednostavan: ne želimo da član tima provede nekoliko sati ponavljajući testiranje interfejsa programa za neke jednostavne operacije poput klikova na linkove, unosa usernamea i passworda i slično. Kako govorimo o web aplikacijama, to znači da ćemo koristiti neki pregledač (Chrome, Firefox, Edge...) da vršimo interakciju sa našom aplikacijom. Selenium upravo radi isto: šalje zahteve ka pregledaču i izvršava radnje kao da je korisnik.

Selenium možemo pokrenuti na operativnim sistemima Windows, Linux i macOS, dok je primena na mobilnim operativnim sistemima moguća, ali iziskuje upotrebu još nekih alata.

Prvi korak je instalacija biblioteke, kao i kod prethodnih biblioteka Pythona, kroz pip, komandom: pip install selenium.

```
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dragoljub Catovic>pip install selenium
Collecting selenium
  Downloading selenium-3.141.0-py2.py3-none-any.whl (904 kB)
    |#####| 904 kB 3.3 MB/s
Requirement already satisfied: urllib3 in c:\users\dragoljub catovic\appdata\local\programs\python\python38-32\lib\site-packages (from selenium) (1.26.4)
Installing collected packages: selenium
Successfully installed selenium-3.141.0
```

Slika 14.2. pip instalacija Seleniuma

Nakon uspešne instalacije biblioteke, možemo odmah i proveriti da li ona radi pravilno. Kreiraćemo jedan novi .py fajl i u njega postaviti sledeći kod:

```
import selenium
print(selenium)
```

Kada pokrenemo fajl, u okviru ispisa možemo videti putanju do selenium foldera, npr.:

```
<module 'selenium' from 'C:\\Users\\User\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages\\selenium\\__init__.py'>
```

Biblioteka sadrži svu potrebnu logiku, ali joj nedostaje ključna komponenta: način da komunicira sa pregledačima na našem računaru. Ovde na scenu stupa Selenium WebDriver.

Selenium WebDriver

Selenium WebDriver prihvata komande poslate iz programskog jezika i šalje ih kao naredbe samom pregledaču. Svaki moderni pregledač ima svoj driver; dakle, važno je koji driver imamo instaliran, koji pregledač koristimo i koje je verzije driver. U okviru ovoga kursa, fokusiraćemo se na Google Chrome i stoga ćemo koristiti njegov driver. Link za preuzimanje web drivera je: <https://chromedriver.chromium.org/downloads>.

Ukoliko budete želeli da koristite Selenium sa drugim pregledačima, na sledećim linkovima možete pronaći drajvere za njih:

Mozilla Firefox: <https://github.com/mozilla/geckodriver>

Microsoft Edge: <https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>

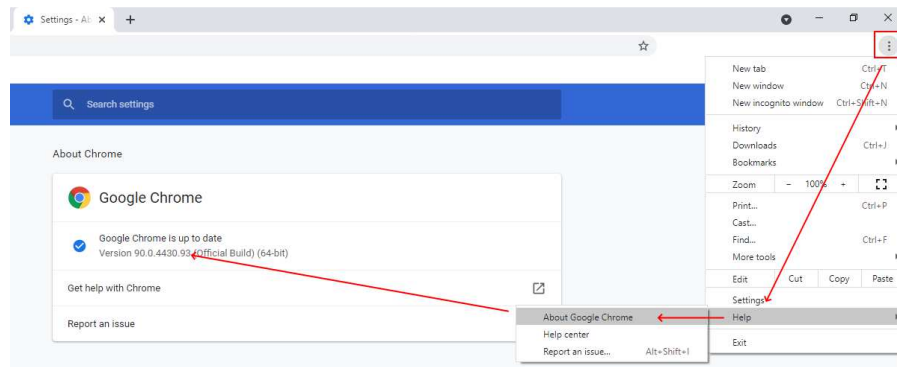
Safari:

https://developer.apple.com/documentation/webkit/testing_with_webdriver_in_safari

Napomena

Verzija WebDrivera koji preuzimate mora biti potpuno identična verziji pregledača kojeg imate instaliranog na računaru.

Pre nego što preuzmemo Chrome WebDriver, potrebno je da imamo, naravno, instaliran Google Chrome pregledač i da proverimo koja je verzija pregledača instalirana. Ovo postićemo pristupanjem podešavanjima pregledača, zatim sekciji Help i opciji About Google Chrome.



Slika 14.3. Provera verzije Google Chromea

Kada smo proverili našu verziju, a pritom nam je bitno samo prvih sedam cifara verzije (u našem slučaju verzija je 90.0.4430), sada preuzimamo WebDriver u istoj verziji. Na sajtu, u okviru sekcije Current Releases biramo verziju 90.0.4430.







Current Releases

- If you are using Chrome version 91, please download [ChromeDriver 91.0.4472.19](#)
- If you are using Chrome version 90, please download [ChromeDriver 90.0.4430.24](#)
- If you are using Chrome version 89, please download [ChromeDriver 89.0.4389.23](#)
- If you are using Chrome version 88, please download [ChromeDriver 88.0.4324.96](#)
- For older version of Chrome, please see below for the version of ChromeDriver that supports it.

Slika 14.4. Izbor verzije Chrome Drivera

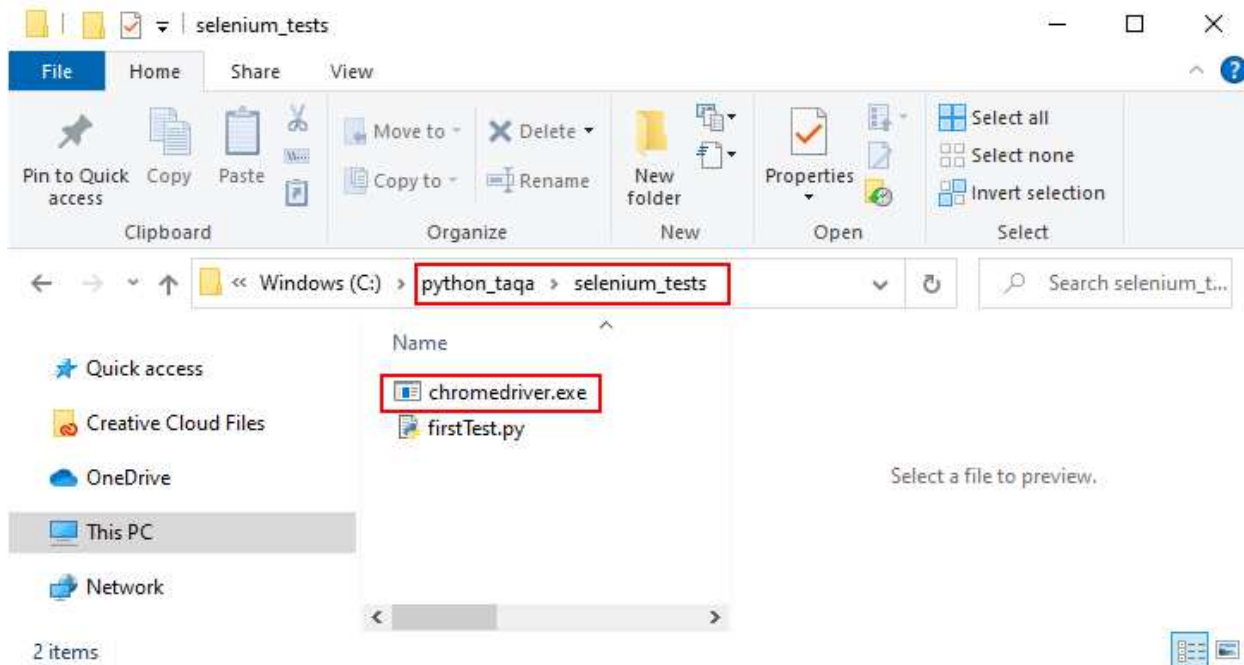
Nakon izbora verzije, u narednom prozoru potrebno je da odaberemo odgovarajući paket za naš operativni sistem. U našem slučaju preuzimamo verziju za operativni sistem Windows.

Index of /90.0.4430.24/

	Name	Last modified	Size	ETag
	Parent Directory	-	-	-
	chromedriver_linux64.zip	2021-03-15 16:49:46	5.53MB	ff32297377308392f3e5b44cf282f77a
	chromedriver_mac64.zip	2021-03-15 16:49:48	7.68MB	01378f44ca91150771859e254809fb66
	chromedriver_mac64_m1.zip	2021-03-15 16:49:50	7.01MB	9cd97b08730a9d395610d051b4aa2c05
	chromedriver_win32.zip	2021-03-15 16:49:51	5.67MB	eeb5e37fc4d4b21337a46576137a2053
	notes.txt	2021-03-15 16:49:56	0.00MB	a79b03d7895fbb145c4d3d0a63ba0d41

Slika 14.5. Izbor instalacionog fajla na osnovu OS-a

Unutar zip fajla se nalazi jedan .exe fajl. Ovaj fajl je potrebno prekopirati na lokaciju gde će se nalaziti Selenium testovi, u našem slučaju .py fajlovi koji će sadržati kod testova. Kako smo već ranije kreirali folder za potrebe našeg kursa, sada možemo kreirati još jedan folder, npr. selenium_tests, i u okviru njega smestiti .exe fajl drivera. Sa ovim smo završili postupak instalacije i unutar ovoga foldera sada možemo kreirati novi .py fajl (primera radi, firstTest.py) koji će sadržati naš Selenium test.



Slika 14.6. Sadržaj novokreiranog Selenium foldera

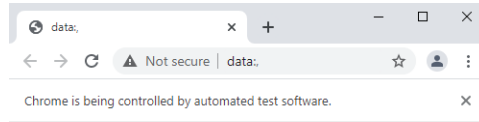
Unutar novokreiranog .py fajla, sada postavljamo kod za Selenium test. Prvi korak je da povežemo naš fajl sa Selenium bibliotekom i web driverom koji smo postavili u folder. Ovo postićemo sledećom linijom:

```
from selenium.webdriver import Chrome
```

Na ovaj način smo definisali da koristimo Chrome pregledač i njegov WebDriver. Sledeći korak je da izvršimo instanciranje pregledača – jednostavnije rečeno, da otvorimo prozor pregledača. Ovo postizemo definisanjem nove promenljive i unutar nje, pozivanjem metode Chrome koja se nalazi unutar WebDrivera:

```
driver = Chrome()
```

Ukoliko sada startujemo ovaj .py fajl, otvoriće se prazan Google Chrome prozor.



Slika 14.7. Otvaranje prozora pregledača od strane Selenium WebDrivera

Ono što je specifično za ovaj, inače standardan prozor pregledača jeste poruka: *Chrome is being controlled by automated test software*. Upravo ova poruka nam sugerise da nismo mi pokrenuli Google Chrome, već je to uradio softver.

Ako dobijate grešku: „'chromedriver' executable needs to be available in the path" odn. .py fajl Vam se ne nalazi u istom folderu kao i ChromeDriver tada je potrebno da prilikom pozivanja metode Chrome postavite putanju do instaliranog Chrome drivera. Npr.

```
driver =  
Chrome("C:/Users/pera/Downloads/chromedriver_win32/chromedriver.exe")
```

Pogledajmo sada kako pomoću WebDrivera možemo da otvorimo neku stranicu unutar Chromea. Ovo postizemo korišćenjem metode get() nad promenljivom driver.

```
driver.get("https://www.link-group.eu/")
```

Kao parametar metode, u vidu stringa prosleđujemo link do neke web stranice. Kada pokrenemo kod, otvara se novi prozor pregledača, koji zatim pristupa adresi koja je navedena u get() metodi.

Naravno, poenta nije samo otvoriti stranicu, već da, kada je dobavimo, nešto dodatno uradimo. Pa recimo da želimo da preuzmемо naslov (title tag) samog sajta. To možemo uraditi na sledeći način:

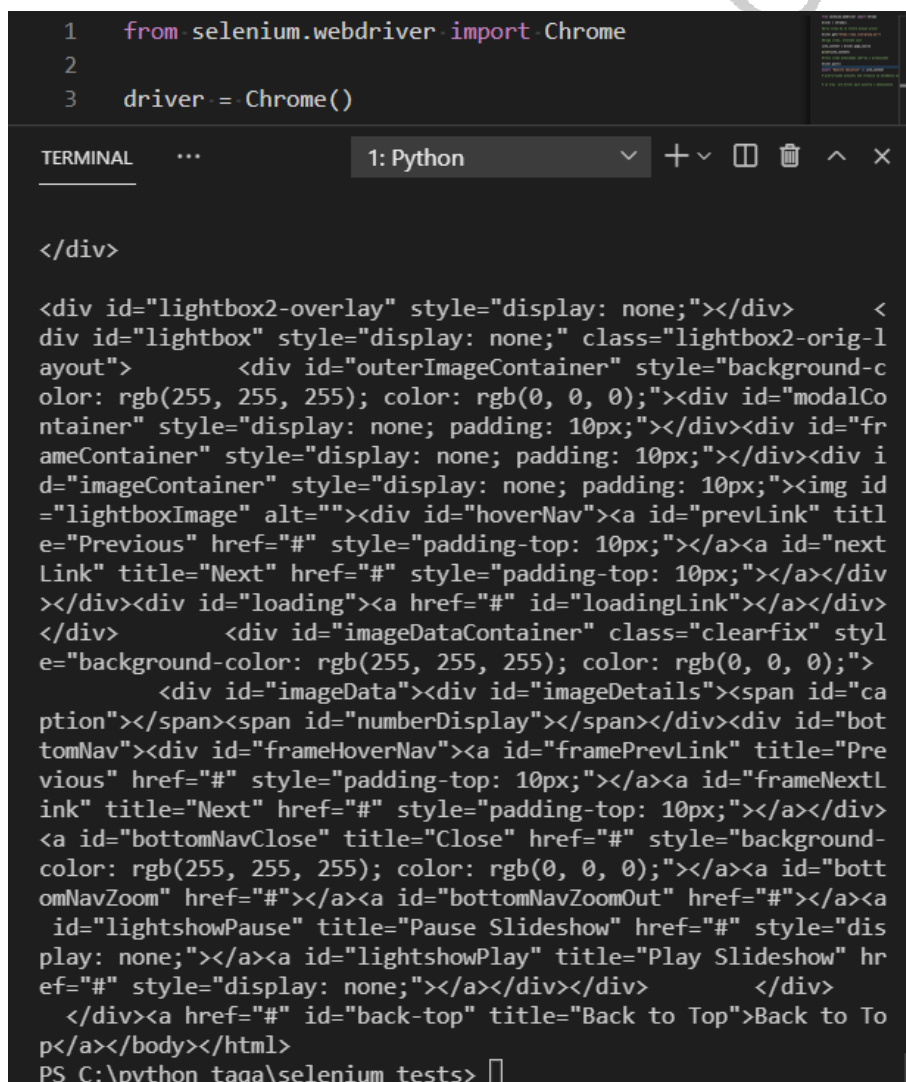
```
from selenium.webdriver import Chrome  
driver = Chrome()  
driver.get("https://www.link-group.eu/")  
print(driver.title)
```

Sada će se u okviru konzole prikazati *LINKgroup*, dakle naslov sajta. Ovo sugeriša da nismo samo otvorili link ka nekom sajtu, već je Selenium preuzeo i kompletan kod, sve aspekte sajta, uključujući i JavaScript kod koji trenutno ne možemo videti. Ovo takođe možemo i proveriti; promenićemo prethodni primer na sledeće:

```
from selenium.webdriver import Chrome
driver = Chrome()

driver.get("https://www.link-group.eu/")
site_content = driver.page_source
print(site_content)
```

Kao što možete videti, kreirali smo novu promenljivu i sada smo nad našom promenljivom driver pozvali parametar `page_source`. Pomoću njega dobavljamo kompletan kod stranice sajta. Nakon toga ispisujemo promenljivu i u konzoli možemo videti prikazan kod stranice.



```
1 from selenium.webdriver import Chrome
2
3 driver = Chrome()

TERMINAL  ...  1: Python  + v  [ ]  [ ]  ^  x

</div>

<div id="lightbox2-overlay" style="display: none;"></div>    <
div id="lightbox" style="display: none;" class="lightbox2-orig-l
ayout">    <div id="outerImageContainer" style="background-c
olor: rgb(255, 255, 255); color: rgb(0, 0, 0);"><div id="modalCo
ntainer" style="display: none; padding: 10px;"></div><div id="fr
ameContainer" style="display: none; padding: 10px;"></div><div i
d="imageContainer" style="display: none; padding: 10px;"><img id
="lightboxImage" alt=""><div id="hoverNav"><a id="prevLink" titl
e="Previous" href="#" style="padding-top: 10px;"></a><a id="next
Link" title="Next" href="#" style="padding-top: 10px;"></a></div
></div><div id="loading"><a href="#" id="loadingLink"></a></div>
</div>    <div id="imageDataContainer" class="clearfix" styl
e="background-color: rgb(255, 255, 255); color: rgb(0, 0, 0);">
    <div id="imageData"><div id="imageDetails"><span id="ca
ption"></span><span id="numberDisplay"></span></div><div id="bot
tomNav"><div id="frameHoverNav"><a id="framePrevLink" title="Pre
vious" href="#" style="padding-top: 10px;"></a><a id="frameNextL
ink" title="Next" href="#" style="padding-top: 10px;"></a></div>
<a id="bottomNavClose" title="Close" href="#" style="background-
color: rgb(255, 255, 255); color: rgb(0, 0, 0);"></a><a id="bott
omNavZoom" href="#"></a><a id="bottomNavZoomOut" href="#"></a><a
id="lightshowPause" title="Pause Slideshow" href="#" style="dis
play: none;"></a><a id="lightshowPlay" title="Play Slideshow" hr
ef="#" style="display: none;"></a></div></div>    </div>
    </div><a href="#" id="back-top" title="Back to Top">Back to To
p</a></body></html>
PS C:\python taqa\selenium tests> [ ]
```

Slika 14.8. Prikaz kompletnog koda stranice

Izmenite kod tako da se ispisuje kompletan kod stranice sajta <https://www.google.com/>.

Do sada ste mogli da primetite da nakon svakog pokretanja prozora Google Chrome pregledača taj prozor ostaje otvoren. Ovo takođe možemo automatizovati; korišćenjem metode `quit()` nad driverom, zatvaramo prozor. Važno je zapamtiti da metodu `quit` postavljamo tek nakon izvršenja potrebnog koda.

```
from selenium.webdriver import Chrome
driver = Chrome()
driver.get("https://www.link-group.eu/")
site_content = driver.page_source
print(site_content)

driver.quit()
```

Pogledajmo sada jedan primer jednog jednostavnog funkcionalnog testa u okviru Seleniuma. Nad prethodnim primerom definisaćemo test slučaj pomoću `assert` naredbe.

```
from selenium.webdriver import Chrome
driver = Chrome()

driver.get("https://www.link-group.eu/")
site_content = driver.page_source
print(site_content)

driver.quit()
assert "Quality education" in site_content
```

Sa ovim `assert`om proveravamo da li se reči *Quality education* pojavljuju u page sourceu, dakle sadržaju našeg sajta. Ovo je, naravno, uprošćen primer, ali princip ostaje isti i ako pretražujemo ceo pasus ili artikal online prodavnice. Na ovaj način, već sada možemo automatizovati proveru sadržaja aplikacije bez naše direktne interakcije sa njom.

Naravno, Selenium sadrži još veći broj opcija. Pre svega, pogledajmo opcije za kontrolisanje samog pregledača, pre nego što nastavimo sa prikazivanjem daljih mogućnosti ove biblioteke.

Ispitati da li se reč Gmail nalazi u okviru stranice <https://www.google.com>.

Postavljanje opcija pregledača

Najčešće se prilikom aktivacije pregledača njemu prosleđuju opcije koje utiču na njegovo ponašanje. U okviru Google Chrome WebDrivera imamo posebnu klasu pod nazivom `ChromeOptions`. Ova klasa se koristi za izmene svojstava samog Google Chromea prilikom pokretanja.

Da bismo koristili klasu, prvo je potrebno povezati je sa našim fajlom; to postizemo linijom:

```
from selenium.webdriver import ChromeOptions
```

Sledeći korak je da definišemo promenljivu koja će postati objekat klase `ChromeOptions`:

```
myOptions = ChromeOptions()
```

Sada smo spremni da dodamo neku dodatnu opciju. U ovu svrhu koristimo metodu `add_argument` nad objektom klase:

```
myOptions.add_argument()
```

Kao parametar metode prosleđujemo naziv opcije koju želimo da dodamo prilikom pokretanja prozora. Recimo da želimo da se prozor pregledača uvek pokreće maksimizovan, dakle preko celog ekrana. U ovu svrhu koristimo parametar `"start-maximized"`:

```
myOptions.add_argument("start-maximized")
```

Naš kod trenutno izgleda ovako:

```
from selenium.webdriver import Chrome
from selenium.webdriver import ChromeOptions

myOptions = ChromeOptions()
myOptions.add_argument("start-maximized")
```

Ukoliko bismo ovo pokrenuli, ništa se ne bi dogodilo jer nemamo definisan driver, dakle sam čin pokretanja pregledača. Možemo odmah dodati i stranicu koju WebDriver treba da pokrene.

```
driver = Chrome()
driver.get("https://www.link-group.eu/")
```

Sada je potrebno da `Chrome()` metodi kažemo da treba da koristi neke posebne opcije prilikom pokretanja. To postizemo tako što prosleđujemo parametar `options` koji će imati vrednosti naše promenljive `myOptions`:

```
from selenium.webdriver import Chrome
from selenium.webdriver import ChromeOptions

myOptions = ChromeOptions()
myOptions.add_argument("start-maximized")

driver = Chrome(options=myOptions)
driver.get("https://www.link-group.eu/")
```

Ukoliko sada pokrenemo kod, videćemo da se otvara maksimizovan prozor pregledača sa željenom stranicom. Pored maksimizovanja prozora, postoji još nekoliko često korišćenih argumenata:

- `incognito` – otvara prozor pregledača u incognito modu;
- `headless` – često korišćen parametar koji sakriva otvaranje prozora;
- `disable-extensions` – prilikom pokretanja isključuje sve ekstenzije;
- `disable-popup-blocking` – isključuje popup prozore.

Otvorite link www.google.com u maksimizovanom prozoru sa isključenim ekstenzijama kao i podešenim zatvaranjem prozora na kraju programa.

Naravno, možemo imati veći broj argumenata prilikom pokretanja – jednostavno dodajemo dodatne pozive metode `add_argument`:

```
from selenium.webdriver import Chrome
from selenium.webdriver import ChromeOptions

myOptions = ChromeOptions()
myOptions.add_argument("start-maximized")
myOptions.add_argument("incognito")

driver = Chrome(options=myOptions)
driver.get("https://www.link-group.eu/")
```

Na ovaj način smo pokrenuli sajt LINKgroup u incognito režimu sa maksimizovanim prozorom.

Navigacija kroz dokument

U okviru Seleniuma postoji mogućnost navigacije kroz dokument. Pod ovim ne mislimo na neku programiranu navigaciju, kao što smo imali u okviru JMetera, već Selenium može da simulira klik na neki element, unos teksta u polje i slično. Kao što smo ranije rekli, poenta Selenium biblioteke je funkcionalno testiranje i ovo je jedan od glavnih načina da automatizujemo testiranje interakcije korisnika sa našom aplikacijom.

U okviru Selenium WebDriver-a imamo više metoda kako da „obeležimo“ element koji treba da bude kliknut, da unesemo tekst u njega i slično.

Pri ovom pristupu, sve se svodi na HTML. Ako se sećate, ranije smo govorili o ID i CLASS atributima jednog HTML elemenata; upravo takve attribute koristimo da označavamo elemente sa kojima Selenium treba da izvrši neku interakciju. U ovu svrhu se koriste WebElement objekti, koji imaju posebne metode za označavanje.

Postoje različite načini za lociranje elemenata na stranici. Selenium obezbeđuje metod **find_element** za lociranje elemenata na stranici.

Da bismo pronašli više elemenata koristimo metodu **find_elements** gde će ona vratiti listu.

Atributi dostupni za klasu `By` se koriste za lociranje elemenata na stranici. `By` klasu možemo importovati linijom: `from selenium.webdriver.common.by import By`.

Atributi dostupni za `By` klasu su:

```
ID = "id"
NAME = "name"
XPATH = "xpath"
LINK_TEXT = "link text"
PARTIAL_LINK_TEXT = "partial link text"
TAG_NAME = "tag name"
```

```
CLASS_NAME = "class name"
CSS_SELECTOR = "css selector"
```

Klasa „By“ služi za određivanje atributa koji se koristi za lociranje elemenata na stranici. Ovo su različiti načini na koje se atributi koriste za lociranje elemenata na stranici:

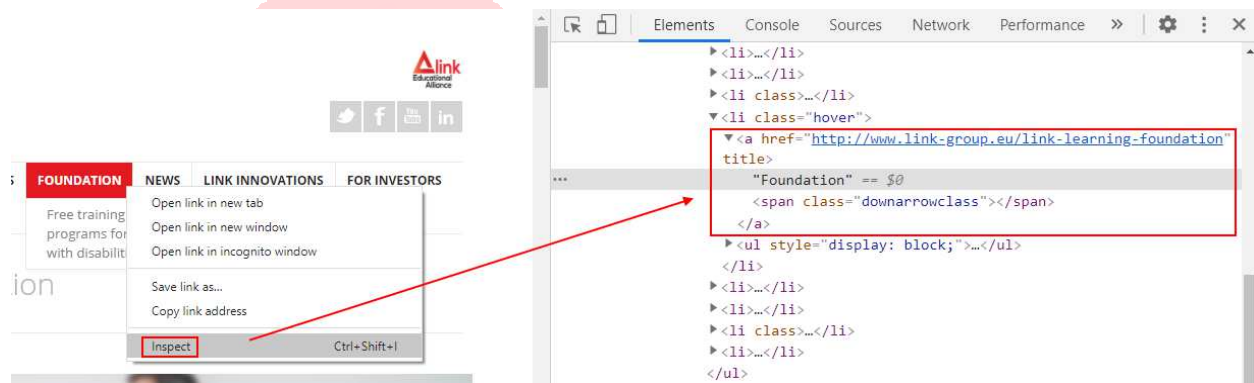
```
find_element(By.ID, "id")
find_element(By.NAME, "name")
find_element(By.XPATH, "xpath")
find_element(By.LINK_TEXT, "link text")
find_element(By.PARTIAL_LINK_TEXT, "partial link text")
find_element(By.TAG_NAME, "tag name")
find_element(By.CLASS_NAME, "class name")
find_element(By.CSS_SELECTOR, "css selector")
```

Ako je potrebno da lociramo nekoliko elemenata sa istim atributom, zamenićemo `find_element` sa `find_elements` metodom.

Pogledajmo sada kako to izvodimo u kodu. Recimo da želimo da na sajtu <https://www.link-group.eu/> kliknemo link ka stranici Foundation. Prvo što nam je potrebno jeste osnova fajla:

```
from selenium.webdriver import Chrome
driver = Chrome()
driver.get("https://www.link-group.eu/")
```

Kako smo rekli da želimo da pristupimo stranici Foundation, treba da utvrdimo koji je najbolji način da označimo samo taj link. Ukoliko iskoristimo ugrađeni Inspect alat pregledača Google Chrome, možemo pogledati koje attribute sadrži element.



Slika 14.9. Inspect alat nad linkom Foundation

Sada je potrebno odlučiti koju metodu koristiti. Kako sam tag nema id, name ili class, ostaju nam dva najbrža načina: korišćenjem metode `find_element(By.LINK_TEXT, "link text")` ili `find_element(By.PARTIAL_LINK_TEXT, "partial link text")`.

Dakle, možemo da koristimo neku od metoda koje će pretražiti sve linkove na stranici i pronaći onaj koji u sebi sadrži tekst FOUNDATION. Obratite pažnju na to da ćemo pretraživati tekst napisan velikim slovima, jer on tako izgleda na sajtu nakon CSS stilizacije.

To u kodu možemo uraditi na sledeći način:

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://www.link-group.eu/")

link = driver.find_element(By.PARTIAL_LINK_TEXT, "FOUNDATION")
```

Dakle, u novoj promenljivoj link čuvamo poziv metode `find_elements_by_partial_link_text` nad objektom klase `Chrome` i kao parametar prosleđujemo tačan tekst samog linka. Da bismo se uverili da smo pronašli samo jedan link, jer ne želimo da kliknemo na više linkova u isto vreme, možemo ispisati vrednost promenljive link.

Pristupite stranici <https://www.youtube.com/> i pronađite link sa definisanim tekstom SIGN IN.

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://www.link-group.eu/")

link = driver.find_element(By.PARTIAL_LINK_TEXT, "FOUNDATION")
print(link)
```

Tada će se u konzoli prikazati nešto poput sledećeg:

```
<selenium.webdriver.remote.webelement.WebElement(session="0c736cfecfe6ae6cc009c2691ee9d89", element="fc337bc5-74e4-41d7-b4c3-196f346991bb")>
```

Ovde možemo videti podatke o elementu koji je Selenium obeležio; takođe, vidimo da je reč o samo jednom elementu. Sada smo spremni za sledeći korak, a to je klik na sam link. Ovo jednostavno postižemo pozivom metode `click()` nad našim link objektom:

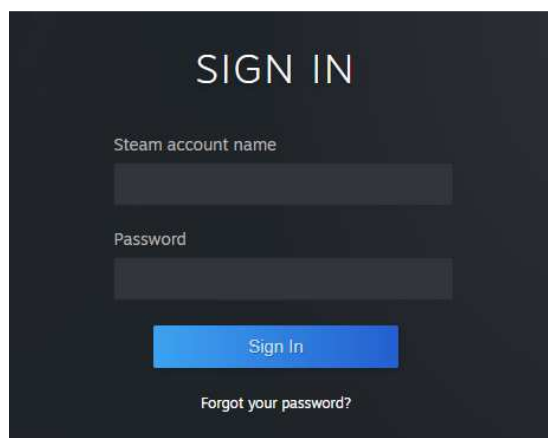
```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://www.link-group.eu/")

link = driver.find_element(By.PARTIAL_LINK_TEXT, "FOUNDATION")
print(link)
link.click()
```

Kada pokrenemo kod, učitaje se stranica i nakon učitavanja automatski će se kliknuti link ka stranici Foundation. Ovde ste imali priliku da vidite pretraživanje elementa prema tekstu. Pogledajmo sada slučaj kada želimo da automatski unosimo tekst u neku formu. Primera radi, iskoristićemo popularnu platformu za distribuciju video igara Steam i njenu login formu, koja se nalazi linku: <https://store.steampowered.com/login>.

Reč je o jednostavnoj formi sa dva polja za unos teksta i jednim dugmetom.



Slika 14.10. Steam login forma

Pristupite stranici <https://store.steampowered.com/> pronadite link sa definisanim tekstom login u gornjem delu stranice i kliknite na njega.

Sada ćemo napisati kod da Selenium automatski unosi podatke i pokuša login u ovu formu. Kao i u prethodnom slučaju, prvenstveno je potrebno da utvrdimo na koji način ćemo obeležiti element. Ukoliko iskoristimo Inspect alat, videćemo da polje za unos korisničkog imena ima jasan id atribut, pod nazivom: `input_username`, dok password polje ima id atribut `input_password`. Ovo nam olakšava izbor metode, jer je najbolja metoda u ovom slučaju metoda `find_element(By.ID, "id")`.

Pogledajmo kako realizujemo označavanje ova dva elementa:

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://store.steampowered.com/login")

usernameField = driver.find_element(By.ID, "input_username")

passwordField = driver.find_element(By.ID, "input_password")
```

Dakle, u nove promenljive smeštamo poziv metode koja označava element prema id-ju i kao parameter navodimo tačne nazive ID atributa HTML elemenata. Sledeći korak je da unesemo tekst u ova polja. Ovo postićemo metodom `send_keys()` koja se poziva nad promenljivama koje čuvaju element. Metodi se prosleđuje tekst koji želimo da unesemo u odgovarajuće polje.

U našem primeru to bi moglo da izgleda ovako:

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://store.steampowered.com/login")

usernameField = driver.find_element(By.ID, "input_username")
passwordField = driver.find_element(By.ID, "input_password")

usernameField.send_keys("johndoe")
passwordField.send_keys("password123")
```

Sa poslednje dve linije koda, prosledili smo username polju vrednost johndoe, dok smo password polju prosledili vrednost password123. Ukoliko pokrenemo naš fajl, videćemo i automatski popunjena polja.

Izmenite kod tako da pristupate stranici <https://www.youtube.com/>, pronađite link sa definisanim tekstom SIGN IN, kliknite na njega i u pronađeno polje za mejl unesite proizvoljni mejl.

Sada nam ostaje samo da kliknemo na *Sign In* dugme forme. Ukoliko inspectujemo kod, videćemo da je ovo dugme najlakše označiti prema tipu elementa, jer je element tipa button na stranici, pa stoga možemo koristiti metodu za označavanje: `find_element_by_tag_name` sa ulaznim parametrom koji će biti naziv tipa elementa. U našem slučaju to je: `button`. Već u narednoj liniji možemo iskoristi sada već poznatu metodu `click` da kliknemo na samo dugme:

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://store.steampowered.com/login")

usernameField = driver.find_element(By.ID, "input_username")
passwordField = driver.find_element(By.ID, "input_password")

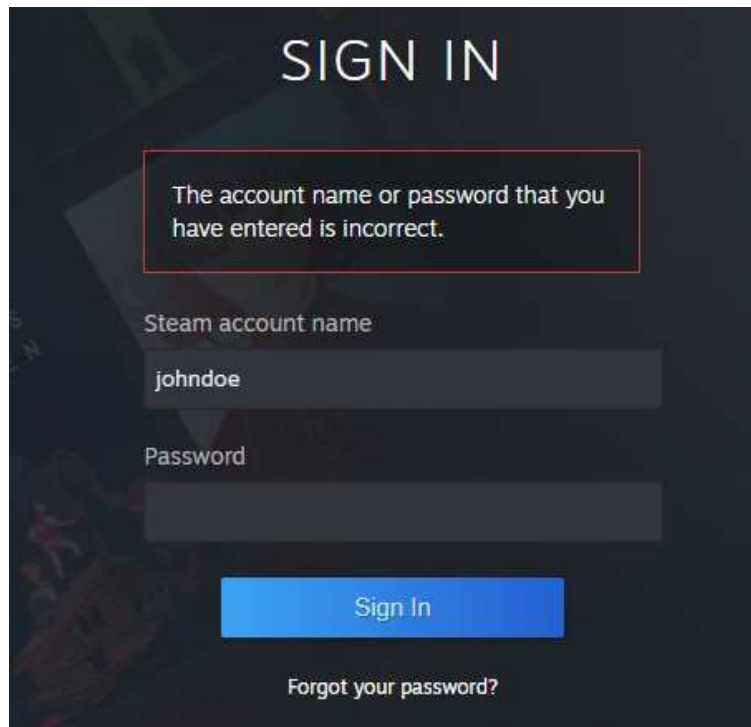
usernameField.send_keys("johndoe")
passwordField.send_keys("password123")

signInButton = driver.find_element(By.TAG_NAME, "button")
signInButton.click()
```

Izmeniti kod tako se u prethodnom primeru posle unosa mejla na stranici logovanja na nalog Youtub-a klikće na polje Next.

Ukoliko sada pokrenemo kod, otvara se stranica, unose se podaci u oba polja i vidimo klik na *Sign In* button, kao i proveru naših podataka i odgovor stranice. Sada se možda pitate – koja je dalja svrha ovoga, osim što možemo da proverimo da li login forma radi? Sada možemo, kao i ranije, da definišemo još test slučajeva. Npr. želimo da proverimo da li se prikazuje poruka o pogrešnom unosu kada unesemo pogrešne podatke. Dakle, da testiramo funkciju naše forme.

Ukoliko pogledamo formu nakon unosa podataka, vidimo da se pojavljuje novi element, koji sadrži tekst greške.



Slika 14.11. Steam login forma nakon unosa

Ukoliko iskoristimo Inspect alat, videćemo da ovaj div element ima ID atribut sa vrednošću: `error_display`. Stoga možemo izvršiti proveru da li se odgovarajući tekst pojavio:

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://store.steampowered.com/login")

usernameField = driver.find_element(By.ID, "input_username")
passwordField = driver.find_element(By.ID, "input_password")

usernameField.send_keys("johndoe")
passwordField.send_keys("password123")

signInButton = driver.find_element(By.TAG_NAME, "button")
signInButton.click()

errorMessage = driver.find_element(By.ID, "error_display")

assert errorMessage.text == "The account name or password that you have
entered is incorrect."
```

Međutim, postoji problem. Nakon izvršavanja koda dobićemo Assertion error iako sve deluje pravilno. Problem je u tome što treba da sačekamo da se forma izvrši pre nego što proverimo da li se tekst pojavio. U okviru Seleniuma imamo metode i za rad sa vremenom, pa je sada pravo vreme da se upoznamo sa njihovom primenom.

Pitanje

Otvaranje linka ka web sajtu u okviru alata Selenium postižemo metodom:

- **get()**
- **set()**
- **open()**

Objašnjenje:

Metodom get() nad driver promenljivom možemo otvoriti neku stranicu unutar pregledača. Kao parametar metode, u vidu stringa prosleđujemo link do neke web stranice.

Čekanje

Postoje dve vrste čekanja u Seleniumu:

- implicitno – gde se čeka određeni vremenski period, određen navedenim parametrom; ukoliko se elementi koji se traže ne pojave u navedenom periodu čekanja, dolazi do greške;
- eksplicitno – kod ovog pristupa se čeka dok se ne ispuni neki uslov.

Implicitno čekanje se vrlo lako realizuje: koristi se metoda `implicitly_wait()`, koja kao parametar uzima vreme u sekundama. Primena ove metode zaustavlja WebDriver i čeka taj period. Primer sintakse upotrebe ove metode bi bio: `driver.implicitly_wait(5)`. Na ovaj način bismo rekli WebDriveru da odloži izvršavanje na pet sekundi.

Naravno, prilikom korišćenja ove metode vodite računa o trajanjima, jer će ona uvek zavisiti od više faktora, a naročito od brzine učitavanja stranica.

Za naš primer implicitna metoda nije pogodna, jer div koji prikazuje grešku postoji od samog učitavanja stranice, samo ne sadrži tekst sve dok se ne unese pogrešna informacija. U ovu svrhu moramo koristiti eksplicitno čekanje, dakle čekanje sve dok se ne ispuni neki uslov. Jedini uslov koji možemo pratiti jeste kada se Sign In dugme može ponovo kliknuti. Ako je dugme spremno za klik, to znači da se forma izvršila i tekst prikazao. Pa pogledajmo kako realizujemo eksplicitno čekanje i definisanje uslova u okviru Selenium testova.

Eksplicitno čekanje

Za kreiranje ovog vida čekanja, koristi se objekat `WebDriverWait` i imamo dve metode: `until` i `until_not`, koje respektivno govore da se čeka dok se neki uslov ne ispuni ili sve dok je neki uslov ispunjen.

Stoga u naš primer moramo, pre svega, importovati tri klase: jedna omogućuje dodavanje logike eksplicitnog čekanja, druga pisanje specifičnih uslova u okviru Seleniuma, a treća alternativnu sintaksu za označavanje elemenata unutar uslova; ove tri linije su:

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.by import By
```

Pogledajmo sada kako pišemo logiku čekanja. Pre svega, u naš kod dodaćemo logiku za definisanje vremena čekanja:

```
wait = WebDriverWait(driver,5)
```

Kao što vidite, u okviru nove promenljive instanciramo klasu WebDriverWait. Ova klasa kao parametre uzima driver koji koristimo i vreme u sekundama. Na ovaj način smo pauzirali rad drivera na pet sekundi. Ali sada želimo da dodamo uslov koji će se čekati. To postizemo prethodno spomenutom metodom until. Metoda until prekida čekanje ukoliko se uslov definisan u okviru nje ispuni. Kako smo rekli da želimo da proverimo da li je dugme ponovo dostupno za klik, korišćemo uslov paketa expected conditions. Pa pogledajmo kako će ta linija izgledati:

```
wait.until(expected_conditions.element_to_be_clickable((By.TAG_NAME,
"button")))
```

Dakle, until metoda uzima samo jedan parametar, a to je uslov. Naš uslov je definisan metodom element_to_be_clickable. Ova metoda će pratiti kada element može da se klikne i na osnovu toga vratiti True ili False. Metoda kao parametar uzima selektor (označavanje elementa). U ovom slučaju koristimo alternativnu sintaksu, tako da možemo sve postaviti kao jedan parametar metode. Pisanje selektora pomoću By klase, ima nekoliko svojih selektora, neke od njih možete videti u nastavku:

Izmeniti primer tako da se postavlja čekanje na 10 sekundi.

Savet je da uvek koristite ovu metodu za selekciju elemenata u okviru expected_conditions metoda Seleniuma.

Sada bi kompletan kod primera izgledao ovako:

```
from selenium.webdriver import Chrome
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.by import By

driver = Chrome()
driver.get("https://store.steampowered.com/login")

usernameField = driver.find_element_by_id("input_username")
passwordField = driver.find_element_by_id("input_password")
```



```
usernameField.send_keys("johndoe")
passwordField.send_keys("password123")

signInButton = driver.find_element(By.TAG_NAME, "button")
signInButton.click()

wait = WebDriverWait(driver,5)

wait.until(expected_conditions.element_to_be_clickable((By.TAG_NAME,
"button"))))

errorMessage = driver.find_element(By.ID,"error_display")
assert errorMessage.text == "The account name or password that you have
entered is incorrect."
```

Ukoliko pokrenemo kod, sve će se pravilno izvršiti i više nećemo imati Assertion error, jer je naša pretpostavka tačna.

Pored prikazanog uslova `element_to_be_clickable`, postoji i veći broj drugih uslova koje možete koristiti u zavisnosti od elementa koji želite da proverite. Kompletan spisak uslova možete pronaći na sledećem linku:

https://www.selenium.dev/selenium/docs/api/py/webdriver_support/selenium.webdriver.support.expected_conditions.html

Dakle, sada smo našim primerom uspešno kreirali funkcionalni test login forme i izvršili dodatno testiranje proverom ishoda rada forme.

Za kraj ove lekcije, pokazaćemo još jednu zanimljivu funkcionalnost alata Selenium. Često se od nas očekuju screenshotovi tokom testiranja. Selenium može generisati screenshot za vas, metodama iz grupe `get_screenshot_as_`. Konkretno, imamo tri metode:

1. `get_screenshot_as_file`
2. `get_screenshot_as_png`
3. `get_screenshot_as_base64`

`get_screenshot_as_file` je najčešće korišćena metoda. Pomoću ove metode, možemo definisati tačnu putanju i format fajla u koji ćemo sačuvati screenshot. Npr.:

`driver.get_screenshot_as_file('form.png')` Na ovaj način, sa izvršavanjem linije, kreiraće se slika pod nazivom `form` i formatom `.png`; slika će se nalaziti pored `.py` fajla unutar foldera projekta.

U svrhu našeg primera, možemo postaviti generisanje screenshota u trenutku kada se pojavljuje poruka o pogrešnom unosu, stoga:

```
from selenium.webdriver import Chrome
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.by import By
```

```

driver = Chrome()
driver.get("https://store.steampowered.com/login")

usernameField = driver.find_element(By.ID,"input_username")
passwordField = driver.find_element(By.ID,"input_password")

usernameField.send_keys("johndoe")
passwordField.send_keys("password123")

signInButton = driver.find_element(By.TAG_NAME, "button")
signInButton.click()

wait = WebDriverWait(driver,5)

wait.until(expected_conditions.element_to_be_clickable((By.TAG_NAME,
"button"))))

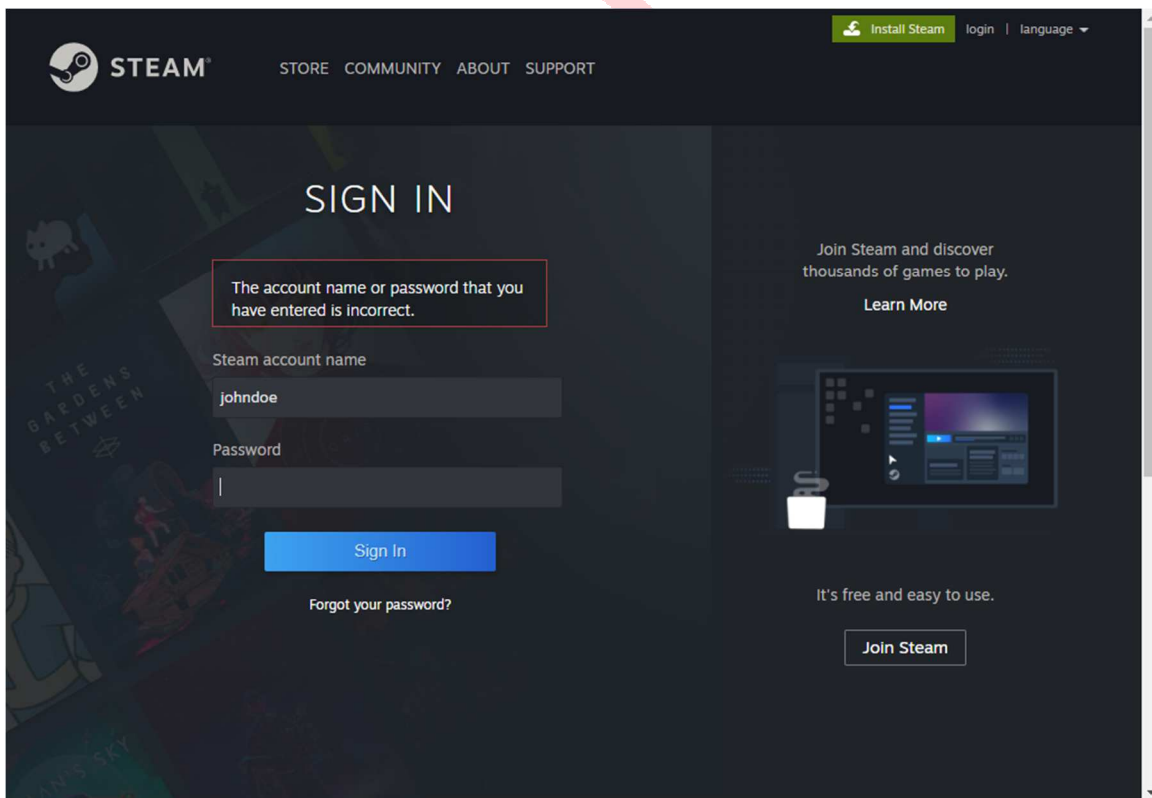
errorMessage = driver.find_element(By.ID,"error_display")

driver.get_screenshot_as_file('form.png')

assert errorMessage.text == "The account name or password that you have
entered is incorrect."

```

Nakon izvršenja koda, u folderu možemo pronaći sliku koja je zabeležila trenutak u kojem se prikazivala greška:



Slika 14.12. Screenshot forme tokom izvršavanja koda

Vežba

Pomoću Seleniuma otvoriti adresu <https://www.google.com/>. U search-u uneti svoje ime i prezime i pokrenuti pretragu. Kliknuti tasteron Enter na search bar i nakon završene pretrage pristupiti linku sa parcijalnim imenom Wikipedia. Uraditi screenshot korišćenjem Seleniuma.

Ukoliko imate nejasnoća u vezi sa izradom vežbe ili želite da proverite svoj kod, na sledećem [linku](#) možete pronaći .py fajl ove vežbe. Napomena: Preuzeti fajl je potrebno postaviti u isti folder gde je i driver za pregledač.

Rezime

- Selenium je softver za upravljanje automatizacijom korišćenja aplikacije. Pruža nam način da simuliramo i automatizujemo kretanje korisnika kroz web aplikacije. Umesto da se operacije obavljaju ručno, koristimo biblioteku koja ovo obavlja za nas i kreiramo sistem da zabeležimo probleme u radu web aplikacije.
- Selenium možemo instalirati kroz pip, komandom: `pip install selenium`. Ključna komponenta koja je potrebna da bi Selenium ostvario komunikaciju sa pregledačima na našem računaru je WebDriver. Svaki moderni pregledač ima svoj driver; dakle, važno je koji driver imamo instaliran, koji pregledač koristimo i koje je verzije driver.
- Metodom `get()` nad driver promenljivom možemo otvoriti neku stranicu unutar pregledača. Kao parametar metode, u vidu stringa prosleđujemo link do neke web stranice. Korišćenjem metode `quit()` nad driverom, zatvaramo prozor pregledača. Važno je zapamtiti da metodu `quit` postavljamo tek nakon izvršenja potrebnog koda.
- Selenium podržava navigaciju kroz dokument: simulaciju klika na neki element, unos teksta u polje i slično.
- Primenom `click()` metode nad selektovanim elementom postizemo simulaciju klika na njega.
- Unos teksta u element postizemo metodom `send_keys()`, koja se poziva nad promenljivom koja čuva selekciju elementa. Metodi se prosleđuje tekst koji želimo da unesemo u odgovarajuće polje.
- U Seleniumu postoje dve vrste čekanja: implicitno i eksplicitno. Kod implicitnog čekanja se čeka određeni vremenski period prema parametru navedenom u sekundama, dok se eksplicitno čekanje vrši sve dok se ne ispuni neki uslov.